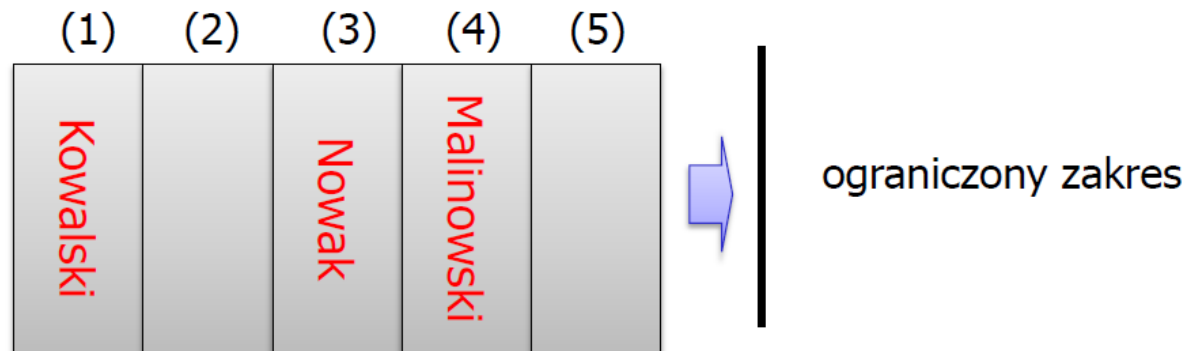


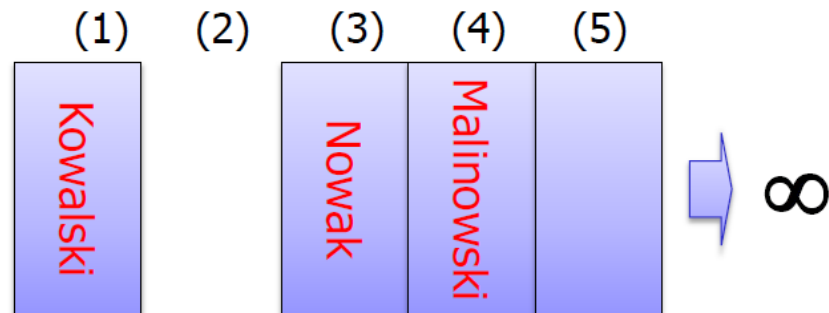
KOLEKCJE

Kolekcje to zbiory obiektów

Tablica o zmiennym rozmiarze (varray)



Tablica zagnieżdżona (nested table)



Porównanie typów kolekcji

własność	VARRAY	NESTED TABLE
maksymalny rozmiar	tak	nie
usuwanie elementów ze środka kolekcji	nie	tak
składowanie w bazie danych	in-line	out-line
zachowanie fizycznego porządku	tak	nie
manipulowanie na pojedynczych elementach kolekcji w SQL	nie	tak

Porównanie typów kolekcji

Kolekcje mogą być tworzone:

- na poziomie SQL - na stałe w bazie danych
- na poziomie PL/SQL - lokalne deklaracje na potrzeby programu

kolekcja typu tabela o zmiennym rozmiarze utworzona z poziomu SQL

```
CREATE TYPE Telefon AS VARRAY(10) OF VARCHAR(15);
```

kolekcja typu tabela o zmiennym rozmiarze utworzona z poziomu PL/SQL

```
DECLARE
    TYPE Telefon IS VARRAY(10) OF VARCHAR(15);
    moi_znajomi Telefon;
BEGIN
    ....
END;
```

Porównanie typów kolekcji

kolekcja typu tabela zagnieżdżona utworzona z poziomu SQL

```
CREATE TYPE Zespol AS TABLE OF Pracownik;
```

kolekcja typu tabela zagnieżdżona utworzona z poziomu PL/SQL

```
DECLARE
    TYPE Zespol IS TABLE OF Pracownik;
    bazy_danych Zespol
BEGIN
    ....
END;
```

Metody kolekcji

metoda	opis
kolekcja(wartość,...)	konstruktor kolekcji, opcjonalnie wstawia wartości jako kolejne elementy kolekcji
EXTEND([n],[i])	rozszerza kolekcję o n pustych elementów, opcjonalnie wypełnia wartością i-tego elementu
TRIM([n])	usuwa n elementów od końca kolekcji
DELETE([n],[m])	usuwa wszystkie elementy kolekcji, n-ty element, lub elementy od n-tego do m-tego
NEXT(n), PRIOR(n)	zwraca indeks elementu następującego (poprzedzającego) elementu o indeksie n
EXISTS(n)	testuje istnienie elementu o indeksie n
FIRST, LAST	zwraca indeks pierwszego (ostatniego) elementu
LIMIT	zwraca maksymalny zakres kolekcji
COUNT	zwraca liczbę elementów kolekcji

Tabela o zmiennym rozmiarze w PL/SQL

```
create or replace type cisnienia as  
varray(5) of numeric(4);
```

```
declare  
    pomiary cisnienia;  
begin  
    pomiary:= cisnienia(995,1020,1009);  
    dbms_output.put_line(pomiary.limit()||' '||pomiary.count());  
    dbms_output.put_line('*****');  
for x in pomiary.first()..pomiary.last() loop  
    dbms_output.put_line(pomiary(x));  
end loop;  
    dbms_output.put_line('*****');  
    pomiary.extend(2,3);  
for x in pomiary.first()..pomiary.last() loop  
    dbms_output.put_line(pomiary(x));  
end loop;  
    dbms_output.put_line('*****');
```

```
5 3  
*****  
  
995  
1020  
1009  
*****  
  
995  
1020  
1009  
1009  
1009  
*****
```

Tabela o zmiennym rozmiarze w PL/SQL

```
create or replace type cisnienia as  
varray(5) of numeric(4);
```

```
pomiary.trim(1);  
for x in pomiary.first()..pomiary.last() loop  
  dbms_output.put_line(pomiary(x));  
end loop;  
dbms_output.put_line('*****');  
pomiary.extend();  
pomiary(5):=1000;  
for x in pomiary.first()..pomiary.last() loop  
  dbms_output.put_line(pomiary(x));  
end loop;  
dbms_output.put_line('*****');  
pomiary.delete();  
dbms_output.put_line( pomiary.count);  
end;
```

995
1020
1009
1009

995
1020
1009
1009
1000

0

Tabela o zmiennym rozmiarze w SQL

Jeśli kolekcja jest tabelą o zmiennym rozmiarze to:

- nie można manipulować pojedynczymi elementami w SQL
- elementy zachowują fizyczny porządek

```
create or replace type DZIENNY_POMIAR as object  
(  
  data_pomiaru DATE,  
  wartosci_pomiarow cisnienia);
```

```
CREATE TABLE POMIARY OF DZIENNY_POMIAR;
```

```
INSERT INTO POMIARY VALUES(DZIENNY_POMIAR(  
  TO_DATE('06/06/2016','DD/MM/YYYY'), CISNIENIA(995,1020,1009)));
```

```
INSERT INTO POMIARY VALUES(DZIENNY_POMIAR(  
  TO_DATE('06/06/2016','DD/MM/YYYY'), CISNIENIA(990,1000)));
```


Tabela o zmiennym rozmiarze w SQL

```
SELECT DATA_POMIARU, WARTOSCI_POMIAROW  
FROM POMIARY;
```

DATA_POMIARU

WARTOSCI_POMIAROW

16/06/06

II153.CISNIENIA(995,1020,1009)

16/06/06

II153.CISNIENIA(990,1000)

Tabela o zmiennym rozmiarze w SQL

```
UPDATE POMIARY
SET WARTOSCI_POMIAROW=CISNIENIA(1100,1001,1015,999)
WHERE
DATA_POMIARU=TO_DATE('06/06/2016','DD/MM/YYYY');
```

```
SELECT DATA_POMIARU, WARTOSCI_POMIAROW
FROM POMIARY;
```

DATA_POMIARU

WARTOSCI_POMIAROW

16/06/06

II153.CISNIENIA(1100,1001,1015,999)

16/06/06

II153.CISNIENIA(1100,1001,1015,999)

Tabela zagnieżdżona w PL/SQL

create or replace type zakupy as
table of character varying (50);

```
declare  
koszyk zakupy;  
x numeric;  
begin  
koszyk:=zakupy('chleb', 'maslo', 'mleko', 'ser');  
koszyk.delete (2,3);  
for x in koszyk.first()..koszyk.last() loop  
if koszyk.exists(x) then  
dbms_output.put_line(koszyk(x));  
end if;  
end loop;  
dbms_output.put_line('*****');
```

```
chleb  
ser  
*****
```

Tabela zagnieżdżona w PL/SQL

create or replace type zakupy as
table of character varying (50);

```
koszyk.delete(1);  
koszyk.extend(2);  
koszyk(5):='smietana';  
koszyk(6):='kasza';  
x:=koszyk.first();  
while x is not null loop  
  dbms_output.put_line(koszyk(x));  
  x:=koszyk.next(x);  
end loop;  
end;
```

```
ser  
smietana  
kasza
```

Tabela zagnieżdżona w SQL

Jeśli kolekcja jest tabelą zagnieżdżoną to:

- można manipulować pojedynczymi elementami w SQL
- należy wskazać tabelę out-line do przechowywania kolekcji

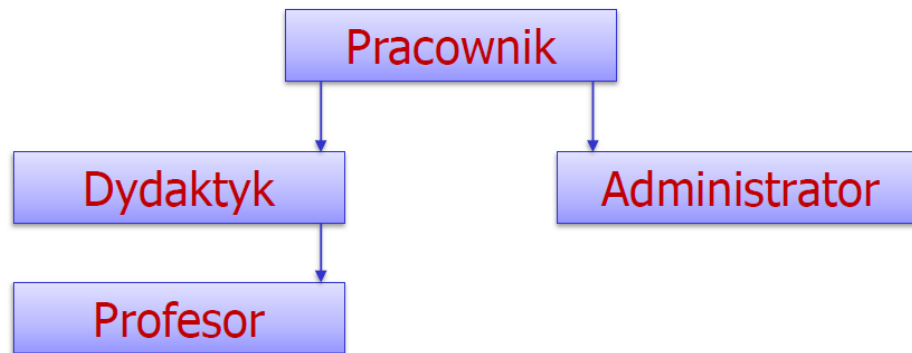
```
create or replace type kupno as object (  
data_kupna date,  
cena numeric(4),  
zawartosc_koszyka zakup);
```

```
CREATE TABLE TOWARY OF KUPNO NESTED TABLE  
ZAWARTOSC_KOSZYKA STORE AS TOWAR_W_KOSZYKU;
```

```
INSERT INTO ZAKUPY VALUES(KUPNO (TO_DATE('06/06/2016',  
'DD/MM/YYYY'), 50, ZAKUP('chleb', 'maslo', 'mleko')));
```

```
INSERT INTO ZAKUPY VALUES(KUPNO(TO_DATE('06/06/2016',  
'DD/MM/YYYY'), 30, ZAKUP('kasza', 'ryz')));
```

Dziedziczenie



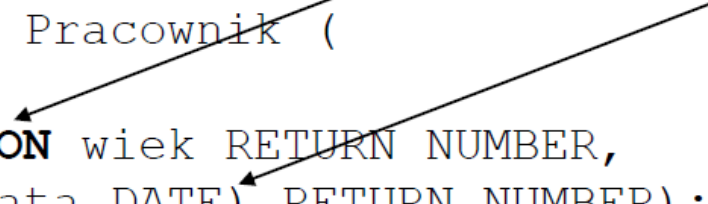
Dziedziczenie polega na definiowaniu nowego typu obiektowego w oparciu o istniejący typ obiektowy

- nowy typ stanowi podtyp (specjalizację) swojego nadtypu (przodka)
- podtyp dziedziczy wszystkie składowe i metody **MEMBER** i **STATIC**
- podtyp może dodawać nowe składowe i przesłaniać metody
- każdy podtyp może mieć tylko jeden nadtyp
- podtyp może dziedziczyć tylko z nadtypu który został zadeklarowany jako **NOT FINAL** (uwaga: domyślnie każdy typ jest **FINAL**)
- metody porządkujące mogą się pojawić tylko w korzeniu hierarchii

Dziedziczenie

```
ALTER TYPE Pracownik NOT FINAL CASCADE;
```

```
CREATE TYPE Dydaktyk UNDER Pracownik (  
    tytul VARCHAR2(10),  
    OVERRIDING MEMBER FUNCTION wiek RETURN NUMBER,  
    MEMBER FUNCTION wiek(l_data DATE) RETURN NUMBER);
```



```
CREATE TYPE BODY Dydaktyk AS  
    OVERRIDING MEMBER FUNCTION wiek RETURN NUMBER IS  
    BEGIN  
        RETURN ROUND(MONTHS_BETWEEN(CURRENT_DATE, data_ur));  
    END wiek;  
  
    MEMBER FUNCTION wiek(l_data DATE) RETURN NUMBER IS  
    BEGIN  
        RETURN EXTRACT(YEAR FROM l_data) -  
            EXTRACT(YEAR FROM data_ur);  
    END wiek;  
END;
```

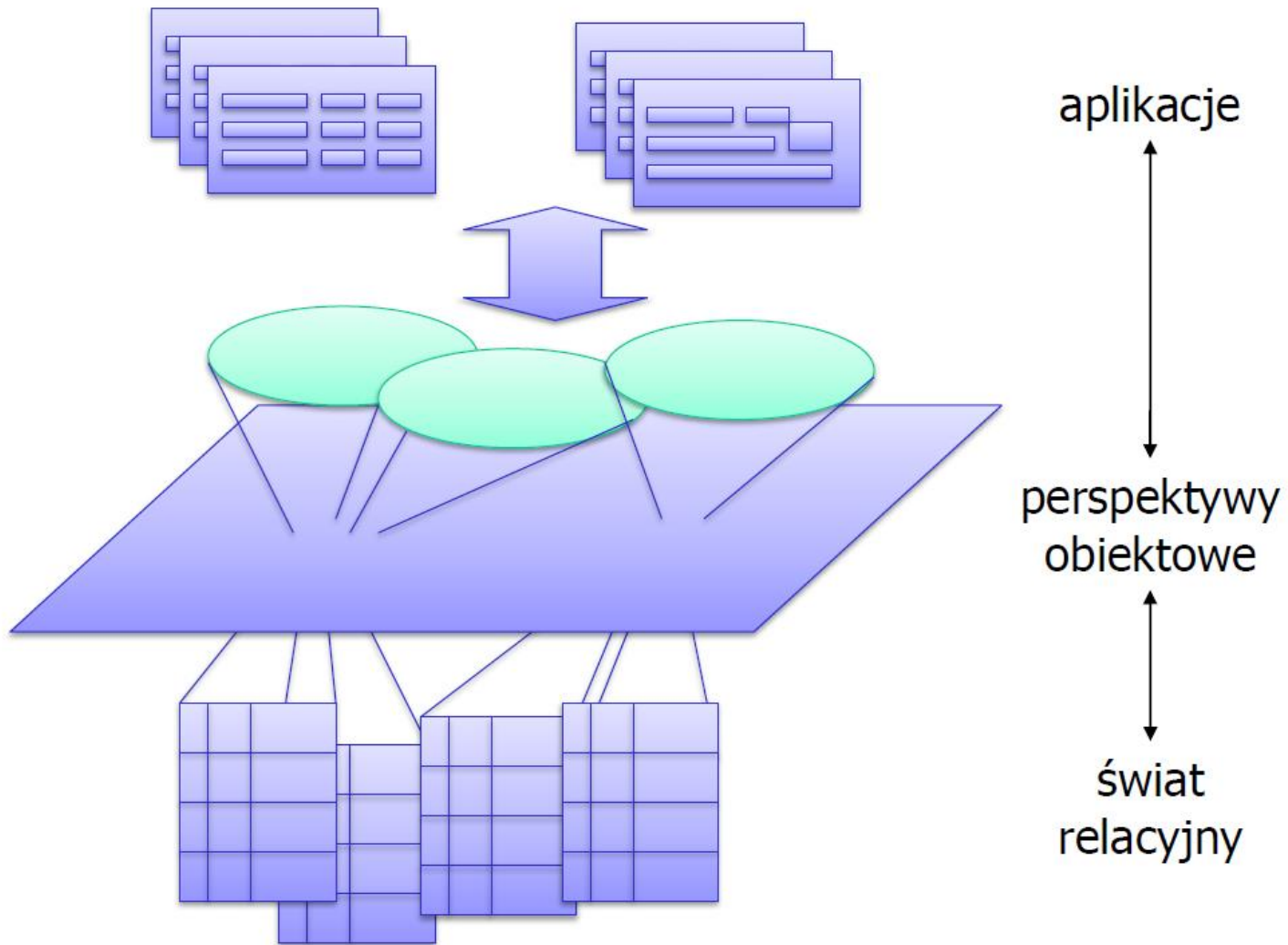
Polimorfizm

Polimorfizm powoduje, że obiekty podtypu mogą zachowywać się jak obiekty swojego nadtypu

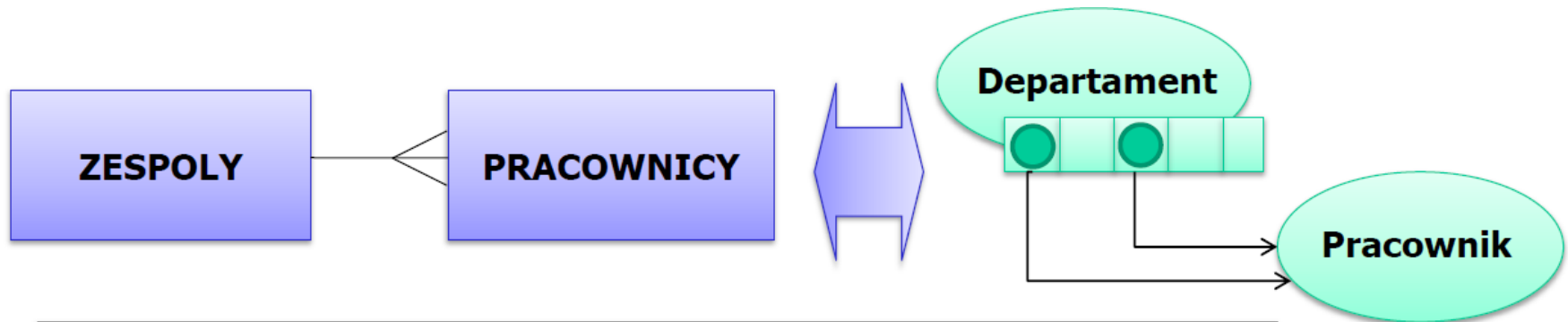
- zamiast obiektu nadtypu można wykorzystać obiekt podtypu
- zamiast obiektu podtypu można wymusić wykorzystanie obiektu nadtypu przez użycie operatora **TREAT**
- wybór wersji metody przeciążonej następuje w momencie wykonania programu (metody są wirtualne)

```
CREATE TABLE pracownicyIdydaktycyObjTab OF PRACOWNIK;  
  
INSERT INTO pracownicyIdydaktycyObjTab VALUES  
(NEW Pracownik('Bolek',2000,'ASYSTENT',DATE '1969-09-03'));  
INSERT INTO pracownicyIdydaktycyObjTab VALUES  
(NEW Dydaktyk('Lolek',5000,'PROF.',DATE '1949-11-13','dr.'));  
  
SELECT p.nazwisko, p.wiek() FROM pracownicyIdydaktycyObjTab p;
```


Perspektywy obiektowe



Perspektywy obiektowe



```
CREATE TYPE PracownicyTab AS TABLE OF Pracownik;

CREATE TYPE Departament AS OBJECT (
  nazwa VARCHAR2(100),
  adres VARCHAR2(100),
  pracownicy PracownicyTab,
  MEMBER FUNCTION iluPracownikow RETURN NUMBER );

CREATE OR REPLACE TYPE BODY Departament AS
  MEMBER FUNCTION iluPracownikow RETURN NUMBER IS
  BEGIN
    RETURN pracownicy.COUNT();
  END iluPracownikow;
END;
```

Perspektywy obiektowe

Utworzenie perspektywy obiektowej

```
CREATE OR REPLACE VIEW DepartamentyObjView OF Departament
WITH OBJECT OID(nazwa) AS
SELECT z.nazwa, z.adres, CAST( MULTISET(
    SELECT NEW Pracownik(nazwisko,placa_pod,etat,zatrudniony)
    FROM pracownicy WHERE id_zesp = z.id_zesp) AS PracownicyTab)
FROM zespoly z;
```

Perspektywy obiektowe

Zapytania

```
SELECT * FROM DepartamentyObjView;
```

```
SELECT d.nazwa, d.adres, d.iluPracownikow()  
FROM DepartamentyObjView d;
```

```
SELECT nazwisko, pensja, etat, data_ur  
FROM TABLE( SELECT pracownicy  
               FROM DepartamentyObjView  
               WHERE nazwa = 'SYSTEMY ROZPROSZONE' );
```